Inverting Step-reduced SHA-1 and MD5 by Parameterized SAT Solvers

Oleg Zaikin
ISDCT SB RAS, Irkutsk, Russia

Abstract

MD5 and SHA-1 are fundamental cryptographic hash functions proposed in 1990s. Given a message of arbitrary finite size, MD5 produces a 128-bit hash in 64 steps, while SHA-1 produces a 160-bit hash in 80 steps. It is computationally infeasible to invert MD5 and SHA-1, i.e. to find a message given a hash. In 2012, 28-step MD5 and 23-step SHA-1 were inverted by CDCL solvers, yet no progress has been made since then. The present paper proposes to construct 31 intermediate inverse problems for any pair of MD5 or SHA-1 steps $(i, i+1)$, such that the first problem is very close to inverting i steps, while the 31st one is almost inverting $i+1$ steps. We constructed SAT encodings of intermediate problems for MD5 and SHA-1, and tuned a CDCL solver on the simplest of them. Then the tuned solver was used to design a parallel Cube-and-Conquer solver which for the first time inverted 29-step MD5 and 24-step SHA-1.

2012 ACM Subject Classification Theory of computation → Constraint and logic programming

Keywords and phrases cryptographic hash function, MD5, SHA-1, preimage attack, SAT, Cube-and-Conquer

Digital Object Identifier 10.4230/LIPIcs..2024.

Funding *Oleg Zaikin*: This study was funded by the Ministry of Education of the Russian Federation, project No. 121041300065-9.

1 Introduction

A cryptographic hash function maps a message of arbitrary finite size to a hash of a fixed size [25]. A secure cryptographic hash function must be resistant to preimage attacks, i.e. it must be computationally infeasible to invert it by finding a message for a given hash. MD5 [37] and SHA-1 [11] are among the most influential and widespread cryptographic hash functions. Given a message, MD5 produces a 128-bit hash. MD5's core component is a compression function that operates on a 128-bit internal state in 64 steps. On each step, the state is modified by mixing with one 32-bit message word. SHA-1 has a similar design, but it produces a 160-bit hash, while its compression function operates in 80 steps. Nowadays both MD5 and SHA-1 are used in practice, e.g. to verify the data integrity. Partially this is because they are still preimage resistant.

It is well known that a cryptographic hash function's resistance can be practically analyzed by algorithms for solving the Boolean satisfiability problem (SAT) [6]. Since it is infeasible to invert MD5 and SHA-1, their weakened versions are usually considered, where some last steps are omitted. In 2007, 26-step MD5 was inverted [10], while for 27- and 28-step MD5 it was done in 2012 [22]. In 2008 and 2012, 22- and 23-step SHA-1 were inverted, respectively [42, 22]. In all these cases it was done via Conflict-Driven Clause Learning [24] (CDCL) solvers. Since 2012, no further progress has been made towards inverting 29-step MD5 or 24-step SHA-1 because the corresponding computational problems are extremely hard. This paper aims to fill these gaps.

When the number of steps of a cryptographic hash function is reduced, the inverse problem can be further simplified by reducing the number of known hash bits [35, 4]. For example, in case of MD5, 29 steps and 64 known hash bits instead of 128 can be considered.

© Oleg Zaikin;
licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, even if this inverse problem is solved, it is not clear if any progress compared to inversion of 28-step MD5 has been made as a result. It is also possible to weaken an inverse problem by partially assigning some bits in a message [8]. However, again, it is unclear whether such weakening contributes to progress or not.

This paper proposes a new type of intermediate inverse problems. Consider an arbitrary cryptographic hash function such that its compression function is divided into steps and on each step a k-bit message word m is mixed with an internal state. Note that most existing cryptographic hash functions, including MD5 and SHA-1, match this condition. Consider a pair $(i, i+1)$ of steps. The idea is to construct $k-1$ intermediate inverse problems by simplifying step $i+1$, while the first i steps are not modified. In the first problem, in step $i+1$ the k-bit message word m is replaced by a k-bit word, where $k-1$ rightmost bits are 0s, while the remaining bit is equal to the leftmost bit in m. In the second problem, $k-2$ rightmost bits are 0s, while 2 remaining bits are equal to that in m and so on. Finally, in the $(k-1)$-th problem, only the rightmost bit in m is replaced by 0. As a result, for a state-of-the-art CDCL solver the first intermediate problem is slightly harder than inverting i steps, then the hardness gradually increases towards inverting $i+1$ steps.

We construct SAT encodings of 31 intermediate inverse problems between 28- and 29-step MD5 for one regular hash. Some of them are solved via the CDCL solver Kissat on a computer in reasonable time. Kissat's parameters are tuned on several simplest problems, and as a result more intermediate problems are solved. A parallel Cube-and-Conquer solver based on the same tuned Kissat inverts 29-step MD5. For SHA-1, its own tuning is performed, and as a result the parallel solver inverts 24 steps.

The paper is organized as follows. Preliminaries on MD5 and SHA-1 are given in Section 2. Intermediate inverse problems are proposed in Section 3. SAT encodings are presented in Section 4. Experiments on the default Kissat are discussed in Section 5. Section 6 presents an algorithm for tuning Kissat. Section 7 describes how Cube-and-Conquer based on the tuned Kissat inverted 24-step SHA-1 and 29-step MD5. Finally, related work is discussed and conclusions are drawn.

2 Preliminaries

This section gives preliminaries on the cryptographic hash functions MD5 and SHA-1.

Cryptographic Hash Function

A *cryptographic hash function* h maps a *message* of arbitrary finite size to a *hash* of finite size [25]. An obligatory property of any cryptographic hash function is that the mapping must be easy to compute, but hard to invert. Consider the following types of resistance.

1. *Collision resistance*: it is infeasible to find any two messages x and x' such that $x \neq x', h(x) = h(x')$.
2. *Preimage resistance*: for any given hash y, it is infeasible to find such a message x' that $h(x') = y$.
3. *Second-preimage resistance*: for any given message x, it is infeasible to find x' such that $x' \neq x, h(x) = h(x')$.

A secure cryptographic hash function must possess all three properties. There are two types of preimage attacks: (i) *practical preimage attack* implies solving an inverse problem, i.e. finding a preimage (message) for a certain hash; (ii) *theoretical preimage attack* is an algorithm for solving an inverse problem with lower complexity than brute force.

The main component of most cryptographic hash functions is a *compression function* that maps an input of fixed size to a shorter output of fixed size. In order to show that such a cryptographic hash function is not secure, it is sufficient to break the resistance of its compression function. In this paper we focus on studying the practical preimage resistance of compression functions of cryptographic hash functions MD5 and SHA-1.

MD5

The Message Digest 5 (MD5) cryptographic hash function was proposed in 1992 [37]. It is a more secure version of MD4 proposed in 1990 [36]. Given a message of arbitrary finite size, *padding* is applied to obtain a message that can be divided into 512-bit blocks. Then a 128-bit hash is produced in accordance with the Merkle-Damgard construction [26, 9], i.e. the compression function is iteratively applied to the blocks.

Given a 512-bit message block, MD5 compression function produces a 128-bit output. The function consists of four rounds, sixteen steps each, and operates by transforming data in four 32-bit registers A, B, C, D. For the first message block, the registers are initialized with the constants specified in the standard. Otherwise, the registers are initialized with an output produced at the previous iteration. The message block is divided into sixteen 32-bit words. In each step, three registers are updated by permuting the current values, while the remaining register is updated by mixing one message word, the current values of all four registers, an additive constant, and a result of the previous step. The mixing is partially done by a round-specific function, while additive constants are step-specific. As a result, all sixteen words take part in each round. When all steps are executed, the registers are incremented by the values they had after the initialization, and the output is produced as a concatenation of A, B, C, D.

Consider a pseudocode of an MD5 step in Algorithm 1. Here $g, 0 \leq g \leq 15$ is a message word index, $1 \leq i \leq 64$ is a step number, $\boxplus$ is addition modulo 2^{32}, and $\lll$ is circular left bit rotation.

Algorithm 1 The i-th step of MD5.

```
**Input**: Current registers' values $A, B, C, D$; step number $i$; message word index $g$; shift amount $s$; round function $Func$.
**Output**: Updated values $A, B, C, D$.
$temp \leftarrow Func(B, C, D) \boxplus A \boxplus K[i-1] \boxplus M[g]$
$A \leftarrow D$
$D \leftarrow C$
$C \leftarrow B$
$B \leftarrow B + (temp \lll s)$
```

Inputs for all 64 steps are specified in [37]. The round functions are as follows: $(x \wedge y) \vee (\neg x \wedge z)$; $(x \wedge z) \vee (y \wedge \neg z)$; $x \oplus y \oplus z$; $y \oplus (x \vee \neg z)$.

In 2004, the first MD5 collisions were published [44]. Regardless, it is still preimage resistant and second-preimage resistant.

SHA-1

The Secure Hash Algorithm 1 (SHA-1) cryptographic hash function was proposed in 1995 as another more secure extension of MD4 [11]. The main differences compared to MD5 are listed below.

1. A 160-bit hash is produced.
2. The compression function operates in 80 steps (4 rounds, 20 steps each) and returns a 160-bit output.
3. Five 32-bit internal registers A, B, C, D, E.
4. Additive constants are round-specific.
5. New round functions: $(x \wedge y) \vee (\neg x \wedge z)$; $x \oplus y \oplus z$; $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$; $x \oplus y \oplus z$.
6. Instead of using parts of the message M directly in transformations, W is used such that $W[t] = M[t]$, $0 \leq t \leq 15$, and $W[t] = (W[t-3] \oplus W[t-8] \oplus W[t-14] \oplus W[t-16]) \lll 1$ if $16 \leq t \leq 79$.

Consider a pseudocode of a SHA-1 step in Algorithm 2.

Algorithm 2 The i-th step of SHA-1.

```
Input: Current registers' values $A, B, C, D, E$; step number $i$; round index $q$; round function $Func$.
Output: Updated values $A, B, C, D, E$.
$temp \leftarrow (A \lll 5) \boxplus Func(B, C, D) \boxplus E \boxplus K[q] \boxplus W[i-1]$
$E \leftarrow D$
$D \leftarrow C$
$C \leftarrow B \lll 30$
$B \leftarrow A$
$A \leftarrow temp$
```

In 2017, the first SHA-1 collisions were published [43], but it is still preimage resistant and second-preimage resistant.

3 Intermediate Inverse Problems for MD5 and SHA-1

This section first proposes a new type of intermediate inverse problems for cryptographic hash functions, then it describes how such problems can be constructed for MD5 and SHA-1.

Intermediate Inverse Problems

Consider an arbitrary cryptographic hash function h such that its compression function f is divided into r steps, and on each step an internal state is mixed with a k-bit word m, where m is either a message word (like in MD5) or a mix of messages' words (like in SHA-1). Besides MD5 and SHA-1, most existing cryptographic hash functions match this condition, e.g. MD4, RIPEMD-160, and SHA-256.

During preliminary experiments, for different triples (cryptographic hash function, SAT encoding, CDCL solver) we attempted to invert modified step-reduced functions, where the last step was weakened in different ways. It turned out that if m is not used, then the last step does not really make the inverse problem harder for the solver. Based on this observation, let us propose the following notation.

Consider i-step and $(i+1)$-step reduced versions of h, where $1 \leq i \leq r-1$. Construct $k-1$ *intermediate step-reduced cryptographic hash functions*, where the first i steps are used as usual, while step $i+1$ is weakened as follows. In the j-th intermediate function, $1 \leq j \leq k-1$, the word m in step $i+1$ is replaced by the word m_{weak} such that the rightmost $k-j$ bits in m_{weak} are equal to 0s, while the remaining j bits are equal to the leftmost j bits in m. Note that this is not the same as replacing m by m_{weak} in the whole h — if m is used in several steps, then the proposed action affects only step $i+1$. In the *j-th intermediate inverse problem* it is needed to find a message given a hash produced by the j-th intermediate step-reduced hash function.

The intention behind this approach is that for a state-of-the-art CDCL solver the first intermediate problem will be slightly harder than the inversion of i steps, then the hardness gradually increases towards the inversion of $i+1$ steps.

Note that if for the j-th intermediate inverse problem between steps i and $i+1$ a preimage is found for a hash, then with probability $\frac{1}{2^{k-j}}$ the preimage inverts the same hash produced by unmodified $i+1$ steps because $k-j$ bits of the corresponding word m can be considered as random bits that can coincide with $k-j$ 0s with the mentioned probability. This is the first take away of the proposed approach — *sometimes it is sufficient to solve an intermediate inverse problem to invert $i+1$ steps.*

Intermediate Inverse Problems for MD5

During preliminary experiments, we tried to weaken step $i+1$ of the MD5 compression function for different values of i by deleting $Func(B,C,D)$ from the sum in the first line of Algorithm 1. However, it did not lead to simpler inverse problems from a state-of-the-art CDCL SAT solver point of view. It was also tried to delete other addends in the first line's sum, to delete the addend B from the fifth line's sum, and to omit shifting in the fifth line. It turned out, that the only action that significantly decreases the hardness is deleting the addend $M[g]$ in the first line. Moreover, if $M[g]$ is deleted, then inverting $i+1$ steps is almost similar to inverting i steps for a CDCL solver.

A pseudocode of an MD5 step weakened according to the proposed idea is presented in Algorithm 3. Here $weakM$ is a 32-bit word, and as a result of two shifts in the first line its rightmost $32-j$ bits are equal to 0, while the remaining j bits are equal to the leftmost j bits in $M[g]$. Then $weakM$ is used instead of $M[g]$. Therefore, to form 31 intermediate inverse problems with increasing hardness, j should be varied from 1 to 31.

Algorithm 3 The $(i+1)$-th weakened step of MD5.

Input: Current registers' values A,B,C,D; step number i; message word index g; shift amount s; round function $Func$; intermediate hash function number j.
Output: Updated values A,B,C,D.

```
1: $weakM \leftarrow (M[g] \gg (32-j)) \ll (32-j)$
2: $temp \leftarrow Func(B,C,D) \boxplus A \boxplus K[i] \boxplus weakM$
3: $A \leftarrow D$
4: $D \leftarrow C$
5: $C \leftarrow B$
6: $B \leftarrow B + (temp \lll s)$
```

In the rest of the paper, the j-th intermediate hash function between MD5 steps i and $i+1$, $1 \leq i \leq 63$, is called *(i j/32)-step MD5*. Note, that according to this notation $j=32$ corresponds to $(i+1)$-step MD5.

Intermediate Inverse Problems for SHA-1

Consider a pair $(i, i+1), 1 \leq i \leq 79$, of SHA-1 steps. Similar to the previous subsection, the idea is to weaken step $i+1$, while the first i steps are used as usual. During preliminary experiments, it was tried to omit operations in Algorithm 2 for different values of i. The picture is the same as for MD5 — when the usage of W is omitted, from the SAT solving point of view the inverse problem becomes the same as the inverse of the first i steps. Yet omitting any other operation, including the usage of the round function, does not make the problem easier. Based on these results, intermediate inverse problems are formed in the same way as for MD5, see Algorithm 4.

Algorithm 4 The $(i+1)$-th weakened step of SHA-1.

```
Input: Current registers' values $A, B, C, D, E$; step number $i$; round index $q$; round function $Func$; intermediate hash function number $j$.
Output: Updated values $A, B, C, D, E$.
$weakW \leftarrow (W[i] \gg (32-j)) \ll (32-j)$
$temp \leftarrow (A \lll 5) \boxplus Func(B, C, D) \boxplus E \boxplus K[q] \boxplus weakW$
$E \leftarrow D$
$D \leftarrow C$
$C \leftarrow B \lll 30$
$B \leftarrow A$
$A \leftarrow temp$
```

4 SAT Encoding

In this study, inverse problems for step-reduced SHA-1 and MD5 compression functions are considered as was done earlier in [22, 35]. This section describes the corresponding SAT encodings.

SHA-1 Encoding

Several SAT encodings of the SHA-1 compression function have been proposed so far [42, 35, 22, 28]. However, it is well known that at the moment the best one is Vegard Nossum's encoding [35]. Compared to the competitors, it produces more compact CNFs which are easier for CDCL solvers [35]. That is why Nossum's encoding has been used in many further studies, e.g. [34, 43].

Recall that in each SHA-1 step 5-ary addition is performed, see Section 2. In the Nossum's encoding, the column addition algorithm is applied, and each column sum is expressed via a pseudo-Boolean constraint which in turn is encoded in the clausal form using the ESPRESSO logic minimizer. Vegard Nossum implemented his encoding in the form of the program SHA1-SAT, which is available online[1]. Given the number of SHA-1 steps and a hash, the program produces the corresponding inverse problem in the CNF form.

For the present study, SHA1-SAT was extended to maintain intermediate inverse problems proposed in Section 3. Assume that $(i+1)$-step SHA-1 is considered, where a combination of message words, the word $W[i]$, is used as an addend, see Section 3, and the corresponding CNF is produced by SHA1-SAT. To encode the j-th intermediate inverse problem between steps i and $i+1$, the following additional actions are performed:

1. An additional 32-bit word $weakW$ is introduced in the form of 32 Boolean variables.
2. The rightmost $32 - j$ bits of $weakW$ are assigned to 0 via adding unit clauses to the CNF.

[1] https://github.com/vegard/sha1-sat

3. The equality conditions for the leftmost j bits of $weakW$ and the corresponding j bits of $W[i]$ are added in the form of $j \times 2$ binary clauses.
4. 32 Boolean variables of $weakW$ are used instead of $W[i]$'s 32 variables in the clauses where the addition is encoded in step $i + 1$.

By using the modified program, CNFs for intermediate inverse problems were constructed between the following pairs of SHA-1 steps: $(21, 22)$, $(22, 23)$, and $(23, 24)$ for 10 hashes: 160 0s, 160 1s, and 8 random ones. Also, for 21, 22, 23, and 24 steps, CNFs for standard inverse problems were generated for the same 10 hashes. In Table 1, the characteristics of CNFs for 23, 23 16/32, and 24 steps are shown.

Table 1 Characteristics of CNFs that encode inverse problems for the main considered step-reduced versions of SHA-1 and MD5.

Hash	Steps	Variables	Clauses	Literals
SHA-1	23	4 288	132 672	873 727
SHA-1	23 16/32	4 480	138 812	913 700
SHA-1	24	4 448	138 764	913 620
MD5	28	7 168	92 520	506 320
MD5	28 16/32	7 424	95 818	524 312
MD5	29	7 392	95 770	524 232

MD5 Encoding

Recently, several SAT encodings of the MD5 compression function have been proposed. The encodings from [10, 22] are not available. The encodings from [40, 46] are available, but we decided to extend SHA1-SAT to support MD5.

In MD5, in each step 4-ary addition is performed, so the corresponding pseudo-Boolean constraint were constructed and encoded in the clausal form using ESPRESSO. As for the round functions (see Section 2), clausal forms of three of them were taken from the SHA-1 encoding, while for the remaining one the clausal form was constructed manually.

CNFs for intermediate inverse problems were constructed between steps $(27, 28)$ and $(28, 29)$ for the same 10 hashes as for SHA-1. In Table 1, the characteristics of main CNFs are shown.

5 Solving Intermediate Inverse Problems by Kissat

First, we decided to study how a state-of-the-art sequential CDCL solver behaves on intermediate inverse problems in case of MD5 and SHA-1. KISSAT of version 3.0 [5] was chosen because this solver and its modifications have showed excellent results in the last three SAT Competitions. Of course, other CDCL solver can be also tried, for example, those which are oriented on cryptanalysis [41, 32, 21]. All described experiments were performed on a PC equipped with the 16-core CPU AMD 3950X and 64 Gb of RAM.

It turned out that inverse problems for 22-step SHA-1 are simple for KISSAT, while for 23 steps the problems are quite hard, but still can be solved in reasonable time. That is why 33 inverse problems were considered: the one for 22 steps, 31 intermediate problems between steps 22 and 23, and the one for 23 steps. As stated in Section 4, we generated 10 instances for each problem — 1 for 160 1s hash, 1 for 160 0s hash, and 8 for random hashes, i.e. 330 instances in total. With the time limit of 24 hours, 303 SHA-1 instances out of 330 were

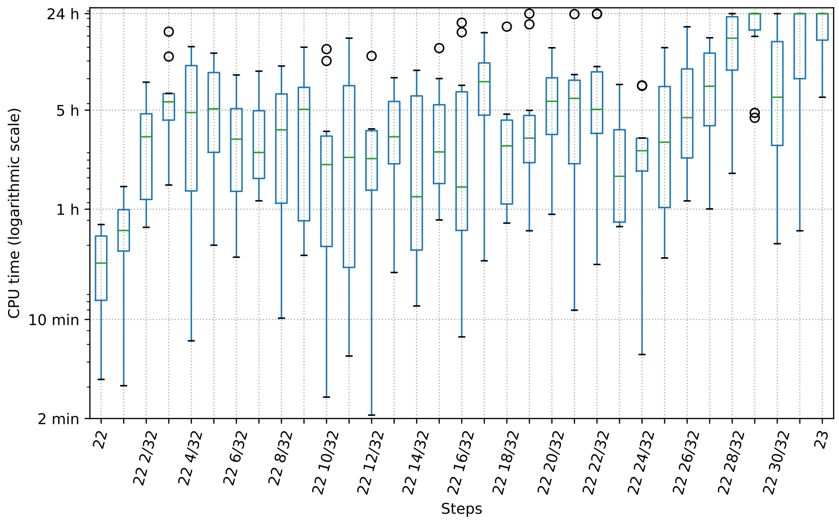

Figure 1 Boxplots of KISSAT3 runtimes on intermediate inverse problems for SHA-1.

solved. Figure 1 shows the runtimes for SHA-1 in the logarithmic scale, where the unsolved instances' values were set to 24 hours. In the figure, the five-number statistics is presented in the form of boxplots: outliers are plotted as small circles, whiskers correspond to the minimum and the maximum (excluding outliers), the median is plotted in green, while a box is plotted from the first quartile to the third quartile.

As for MD5, 330 instances between steps 27 and 28 were generated in the same way. With the time limit of 24 hours, all MD5 instances were solved, see Figure 2.

According to the results, in case of MD5 the hardness of the intermediate inverse problems grows almost linearly. In case of SHA-1 there is a clear leap between steps 22 1/32 and 22 2/32, then the hardness remains more or less the same, and finally the second leap happens between steps 22 27/32 and 22 28/32. Note, that 24 out of 27 unsolved instances were parts of the last 5 series, i.e. starting from 22 28/32 steps. Similar experiments were held between steps 21-22 of SHA-1 and 26-27 of MD5 and the patterns were the same there.

Based on these results, the second take away of the approach, proposed in Section 3, is formulated: *a runtime estimation for an unreachable step can be calculated by extrapolating runtimes of the previous step and (some) intermediate problems.*

As mentioned in Section 2, it is sufficient to invert any hash to break the preimage resistance of a cryptographic hash function. However, in practice this should be a regular hash, say all 0s or all 1s, otherwise it may be hard to justify this choice and prove that it was not done for a random pair of input and output. In this paper, 128 1s and 160 1s are chosen for MD5 and SHA-1, respectively. Further they are called *1-hashes*. Instances of intermediate inverse problems between steps 28-29 of MD5 and steps 23-24 of SHA-1 were generated for 1-hashes. Recall that at most 28-step MD5 and 23-step SHA-1 are inverted in the literature. Out of 31 intermediate problems, 6 were solved by KISSAT within the time limit of 24 hours in case of MD5. In particular, the intermediate problems with j of 1, 3, 5, 8, 9, and 10 were solved. Yet only one intermediate problem with $j = 20$ was solved for SHA-1. It means that (28 10/32)-step MD5 and (23 20/32)-step SHA-1 are inverted, and this is already a clear progress compared to the literature, but in the next sections we are going further.

The third take away is as follows: *by solving intermediate inverse problems, some progress*

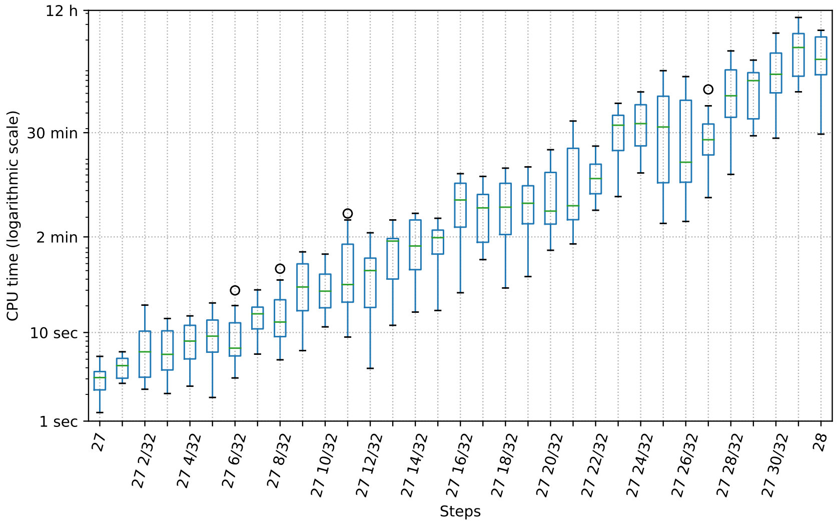

Figure 2 Boxplots of Kissat3 runtimes on intermediate inverse problems for MD5.

can be achieved compared to the state of the art.

6 Parameterization Algorithm

A hypothesis behind the proposed method is that simple intermediate inverse problems can be leveraged to parameterize Kissat (or, in other words, tune its parameters' values) to solve hard intermediate problems faster or just solve within the time limit if it is not yet so. On the one hand, the tuning simplifies problems which are already simple, and it is not guaranteed that it will help solving hard problems. On the other hand, all corresponding CNFs have a very similar structure. We decided to test this hypothesis in practice.

Parameterization of SAT solvers is a well-developed field, see [18]. Formally, the general problem is to automatically identify a *parameter configuration* (a set of parameters' values) that maximizes the performance of a SAT solver across a set of instances [18]. Among the main parameterization algorithms for SAT are: ParamILS [20], SMAC [19], and GGA [2]. A recent implementation of GGA is PyDGGA [1] while that for SMAC is SMAC3 [23]. In this study, for this purpose an extension of a simple yet powerful (1+1) evolutionary optimization algorithm ((1+1)-EA [30]) is used to have a full control over the tuning process. Another reason is that evolutionary optimization algorithms have been successfully applied recently to speed up parallel SAT solving [39, 47].

In short, (1+1)-EA works as follows: a Boolean vector of size k is given, and a series of k independent Bernoulli trials [29] with the success probability $1/k$ is performed. If $i \in \{0, \ldots, k-1\}$ corresponds to a successful trial, then the i-th value is flipped. The obtained vector is compared with the best one via an objective function, then a new vector is generated and so on. It is clear that only binary parameters can be tuned via (1+1)-EA.

We propose an integer extension of (1+1)-EA that is further called (1+1)-EA-Int. The pseudocode is presented in Algorithm 5. This algorithm operates with parameters of arbitrary finite sizes by applying the categorical distribution — a discrete probability distribution that describes the possible results of a random variable that can take on one of n possible categories, each with specified probability [31]. Assume that P is a set of parameters values. To choose a new value for the i-th parameter, the function $categ\_dist(x_i, P_i)$ is called, where x_i is the current value of the i-th parameter and $P_i, |P_i| = k$ is a set of possible values of the i-th parameter. In this function, a weights vector w of length k is formed. First, w_i is assigned to 0. Assume that $maxdist = max(i-1, k-i-2)$, then $w_{i-1} = w_{i+1} = 2^{maxdist}$, $w_{i-2} = w_{i+2} = 2^{maxdist-1}$ and so on. As a result, at least one of two elements w_0, w_{k-1} becomes 1. Finally, the probabilities are assigned to parameters values as follows: $(\frac{w_0}{\sum_{j=0}^{k-1} w_j}, \frac{w_1}{\sum_{j=0}^{k-1} w_j}, \ldots, \frac{w_{k-1}}{\sum_{j=0}^{k-1} w_j})$. The idea is that the closer a value to the current one, the higher the probability is, yet any value (except the current one) can be chosen with non-zero probability.

Algorithm 5 (1+1)-EA-Int.

Input: parameters values P, a start configuration x^0, an objective function f.
Output: the best found configuration x^{best} with the objective function's value f^{best}.

```
$n \leftarrow param\_num(x^0)$
$\langle x^{best}, f^{best} \rangle \leftarrow \langle x^0, f(x^0) \rangle$
$checked\_conf \leftarrow \{\}$
while not termination criteria do
    $x^{cur} \leftarrow x^{best}$
    for $i \leftarrow 0$ to $n-1$ do
        with probability $\frac{1}{n}$ $x_i^{cur} \leftarrow categ\_dist(x_i^{cur}, P_i)$.
    if $x^{cur}$ in $checked\_conf$ then
        continue
    $f^{cur} \leftarrow f(x^{cur})$
    $checked\_conf.add(x^{cur})$
    if $f^{cur} < f^{best}$ then
        $\langle x^{best}, f^{best} \rangle \leftarrow \langle x^{cur}, f^{cur} \rangle$
return $\langle x^{best}, f^{best} \rangle$
```

Consider an example. Assume that a parameter has possible values $1, 2, 5, 10, 25$, while its current value is 10. Since $maxdist = max(3-1, 5-3-2) = 2$, it follows that $w = (1, 2, 4, 0, 4)$, and the probabilities are $(\frac{1}{11}, \frac{2}{11}, \frac{4}{11}, 0, \frac{4}{11})$.

Note, that additionally in Algorithm 5 all checked configurations are remembered to avoid redundant calculations since the objective function is costly.

In all experiments, given a sample of CNFs and a parameter configuration, the objective function f is the sum of Kissat's runtimes on the CNFs, when the configuration is applied to the solver. The goal is to minimize the function on the space of parameter configurations.

7 Inverting by Parameterized Solvers

In this section, the parameterization algorithm proposed in Section 6 is applied to tune Kissat on simple intermediate inverse problems to invert 24-step SHA-1 and 29-step MD5.

Experimental Setup

We implemented the parameterization algorithm from Section 6 in Python, where the objective function is calculated in parallel — 1 configuration per CPU core. The implementation is available online[2]. The extended version of sha1-sat (see Section 4) and all generated CNFs

[2] https://github.com/olegzaikin/paramsat

are available online as well[3]. The same PC as in Section 5 was used in the experiments. The termination criteria were as follows: at most 24 (or 72 in some cases) hours and 1 000 calculations of the objective function.

There are 90 parameters in KISSAT 3.0, yet 50 of them were excluded from the consideration since in preliminary experiments they did not affect the performance. The remaining 40 parameters are presented in Table 2. All these parameters are integer, and their ranges are specified in the solver's documentation [5]. As a start configuration x^0 in Algorithm 5 the default KISSAT's configuration was leveraged.

Table 2 Varied parameters of KISSAT.

backbone	eliminateclslim	reluctantlim	sweepdepth
backbonerounds	eliminateocclim	restartint	sweepfliprounds
bumpreasonslimit	eliminaterounds	restartmargin	sweepmaxclauses
bumpreasonsrate	emafast	shrink	sweepmaxdepth
chronolevels	emaslow	stable	sweepvars
compactlim	mineffort	substituteeffort	target
decay	minimizedepth	substituterounds	tier1
definitioncores	modeinit	subsumeclslim	tier2
definitionticks	reducefraction	subsumeocclim	vivifytier1
eliminatebound	reluctantint	sweepclauses	vivifytier2

Inverting 24-step SHA-1

In preliminary experiments it turned out that a training set must contain at least 10 CNFs to obtain a configuration that works better on harder intermediate inverse problems. For SHA-1, the training set consisted of 16 CNFs: the last 15 intermediate inverse problems between steps 21-22 and inverting 22-step SHA-1, all for 1-hash. These CNFs were chosen because the objective function's value on them is reasonable: 1 hour 58 minutes on 1 CPU core. For comparison, on the first 16 intermediate inverse problems between steps 22-23 the corresponding value is 44 hours.

The parameterization algorithm was run three times with seeds 0, 1, and 2 and the time limit of 24 hours. It stopped after 11, 12, and 16 hours, respectively, because the limit on the objective function calculations (1 000) had been reached. The best result was found with seed of 1: the best configuration was updated 13 times, and on the final one the objective function's value was 22 minutes (compared to 1 hour 58 minutes on the default configuration). Table 3 contains those 12 parameters which changed their values in the found configuration.

The tuned and the default KISSAT3 were run with the time limit of 24 hours on 64 CNFs: 31 intermediate ones between steps 22-23, 1 for 23 steps, 31 intermediate ones between steps 23-24, and 1 for 24 steps. The results are presented as a cactus plot in Figure 3. The default version solved 30 intermediate problems between steps 22-23, while the tuned version solved all 31 of them. Both versions inverted 23 steps. Out of 31 intermediate problems between steps 23-24, the default version solved 1 problem, while the tuned one coped with 5 problems. All in all, the tuned version solved 5 more problems, and on most problems it was significantly faster.

[3] https://github.com/olegzaikin/sha1-sat

Table 3 The best KISSAT's configuration found for SHA-1.

Parameter	Default value	Found value
backbonerounds	100	10
definitionticks	1 000 000	100
eliminatebound	16	32
eliminateclslim	100	10
emafast	33	10
minimizedepth	1 000	100
restartmargin	10	20
stable	1	2
sweepfliprounds	1	5
sweepmaxclauses	4 096	2 147 483 647
sweepvars	128	64
vivifytier1	3	2

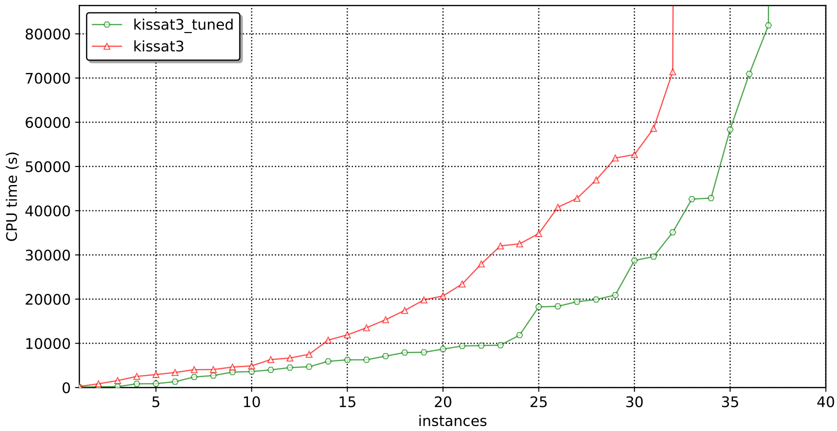

Figure 3 Comparison of the default KISSAT with its tuned version on intermediate inverse problems for steps 22-24 of SHA-1, 1-hash.

According to Figure 3, the hardness of the intermediate inverse problems increases quite smoothly in case of SHA-1, that is why it was decided not to run any additional tuning since 5 out of 31 intermediate inverse problems between steps 23-24 were already solved by the tuned solver. To invert 24-step SHA-1, experiments were run on the supercomputer "Akademik V.M. Matrosov"[4]. Each supercomputer's node is equipped with a 36-core CPU and 128 Gb RAM. At most 5 nodes (180 cores) were taken for one task.

To parallelize hard intermediate problems, the Cube-and-Conquer [13] approach was applied, where a given problem is split via lookahead into subproblems, which are solved by a CDCL solver. The lookahead solver MARCH_CU [14] was used to split the inverse problem for 24-step SHA-1 into 166 subproblems via the cutoff parameter $n=3467$. Then the same tuned KISSAT was run on the subproblems in the form of a 167-core task with the time limit of 3 days. For this purpose, the MPI program CONQUER_MPI was run[5], which used 1 core

[4] Irkutsk Supercomputer Center of SB RAS, http://hpc.icc.ru

[5] https://github.com/olegzaikin/EnCnC

for the master process and 166 cores for computing processes. Table 4 presents a solution that was found in 23 hours. The default Kissat was also run on the same subproblems on the supercomputer, yet no solution was found within 3 days.

Table 4 A preimage of 160 1s produced by 24-step SHA-1.

`0xa6c5c463`	`0x182655e0`	`0x2c5ba5f0`	`0xe0028033`
`0x8c3779b1`	`0x98635880`	`0xc5b822e`	`0x297efce7`
`0x59987038`	`0xd764eca9`	`0x7ed9801d`	`0xdde4f1e0`
`0x524e678`	`0xa8ce47dc`	`0xa813fd76`	`0x8b58e09f`

Inverting 29-step MD5

For MD5 as a training set the first 16 intermediate inverse problems between steps 27-28 were chosen. The parameterization algorithm was run with seeds 0, 1, and 2 with the time limit of 24 hours, and it stopped after 4, 3, and 2 hours, respectively. The best result was achieved on seed 0: the best configuration was updated 8 times, and on the final one the objective function's value was 4 minutes (compared to 14 minutes on the default configuration).

The tuning was run one more time starting from the found configuration. This time the training set contained the last 15 intermediate problems between steps 27-28, and the problem for 28 steps. The reason was that on these 16 CNFs on the found configuration the objective function value was 5 hours 27 minutes, that is still reasonable, while on the default configuration it was 18 hours 37 minutes. The time limit was increased to 3 days, and the algorithm was run three times with seeds 3, 4, and 5. In all cases the time limit was reached: the objective function was calculated 584, 698, and 973 times, respectively. The best result was found with seed 5: the best configuration was updated 8 times, and the final objective function's value was 1 hour 30 minutes. Table 5 contains those 16 parameters which changed their values in the found configuration compared to the default one.

Table 5 The best Kissat's configuration found for MD5.

Parameter	Default value	Found value
chronolevels	100	1 000
decay	50	32
definitionticks	1 000 000	100
eliminatebound	16	2
eliminateocclim	2 000	1 000
emaslow	100 000	75 000
minimizedepth	1 000	100
restartmargin	10	20
shrink	3	0
stable	1	2
substituterounds	2	32
subsumeclslim	1 000	10 000
sweepmaxclauses	4 096	2 048
target	1	2
tier2	6	10
vivifytier2	6	5

The tuned and the default Kissat3 were run on the PC with the time limit of 24 hours on 32 CNFs: 31 intermediate inverse problems between steps 28-29, and the inverse problem for 29 steps. The results are presented as a cactus plot in Figure 4. The default version solved 6 intermediate problems for j of 1, 3, 5, 8, 9, and 10, while the tuned one solved 10 problems for j of 1, 2, 3, 5, 6, 7, 8, 9, 11, and 13. As a result, (28 13/32)-step MD5 was inverted. The figure clearly shows that the tuned version's performance is much better.

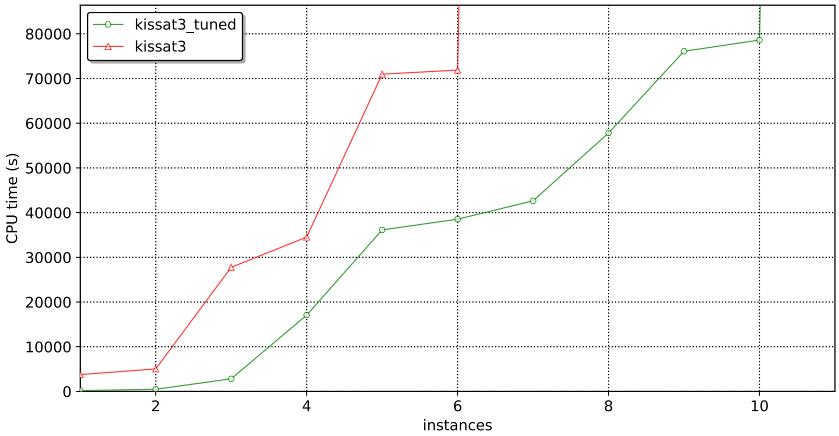

Figure 4 Comparison of the default Kissat with its tuned version on intermediate inverse problems for steps 28-29 of MD5, 1-hash.

On the next stage, intermediate inverse problems with j of 20, 24, and 28 between steps 28-29, as well as the inverse problem for 29 steps were considered. For each problem, at most 180 cubes were generated, a 180-core task was formed, and the tuned Kissat was run on each subproblem with the time limit of 24 hours on the supercomputer. On the first two problems solutions were found in 4 and 7 hours, respectively, yet nothing was found for the remaining two ones. It became clear that the inverse problem for 29 steps is quite hard, so this problem was split via march_cu into 74 470 cubes using the threshold $n = 5493$. These cubes were divided into 10 parts to form 10 7-days 180-core tasks with the time limit of 15 000 seconds for each subproblem. The first two task were completed with no found solution, yet in the third task a solution was found in 37 hours. The corresponding preimage is presented in Table 6.

Table 6 A preimage of 128 1s produced by 29-step MD5-1.

0xe1051a9e	0x48120773	0x996a5457	0xaaa1d815
0x37d8149c	0x5f999c05	0x182ba14b	0xdfff1673
0xc5db0a2f	0x44430b2a	0xa269f5a2	0x69781b85
0x2b7f0939	0xc1ff3c22	0xc55e990f	0x96ba3fb8

Discussion

According to the results, the hypothesis proposed in Section 6 is experimentally confirmed, and it gives the fourth take away: *tuning a state-of-the-art CDCL solver's parameters on simple intermediate inverse problems allows faster solving of hard intermediate inverse problems.*

Two found configurations for SHA-1 and MD5 differ quite a lot, but the following values exist in both of them: definitionticks=100, minimizedepth=100, restartmargin=20, and stable=2.

It seems realistic to invert 25-step SHA-1 via the proposed approach, but for 30-step MD5 it is likely that additional effective algorithmic techniques are required.

Reproducibility

The preimages were hard to find, but their verification via direct computations takes a fraction of a second. The correctness in case of 24-step SHA-1 was verified via the tool VERIFY-PREIMAGE from Nossum's repository SHA1-SAT. As for MD5, it was done by modifying the reference implementation from [37] written in C. First, padding (see Section 2), as well as all steps but the first 29 ones must be deleted. Then the preimage should be given as an input. As a result, 128 1s are produced.

8 Related Work

SAT-based cryptanalysis has been applied to cryptographic hash functions of the MD family as follows. In [27], practical collision attacks on MD4 and MD5 were performed. In [10], 39-step MD4 was inverted, while for steps 40-43 it was done in [45]. 26-step MD5 was inverted in [10], while for 27- and 28-step versions it was done in [22, 46]. As for the SHA family, the situation is as follows. Step-reduced versions of SHA-0, SHA-256, and SHA-3 were inverted in [22, 17, 34]. 22- and 23-step SHA-1 were inverted in [42, 22]. In [4], a weakened 24-step SHA-1 was inverted, such that the number of known hash bits was reduced from 160 to 128. In [35], full-step version of SHA-1 was inverted, but most message bits were assigned randomly, yet in [8] a similar result was achieved for SHA-256. Algebraic fault attacks on SHA-1 and SHA-256 were proposed in [33]. Collisions for SHA-1 were found in [43].

The aforementioned SAT-based preimage attacks on step-reduced MD5 and SHA-1 are practical. Also, theoretical preimage attacks on 62-step SHA-1 and full MD5 exist [12, 38].

Recently, Cube-and-Conquer has been successfully used to solve the Boolean Pythagorean Triples problem [16], the Schur number five problem [15], Lam's problem [7], and in model finding [3].

9 Conclusions and Future Work

This paper proposed a new type of intermediate inverse problems between any two steps $(i, i+1)$ of a cryptographic hash function from a wide class. First, these problems are useful to make some progress if i steps can be inverted in reasonable time while the inversion of $i+1$ steps is infeasible. Second, the simplest intermediate problems can be used to tune a CDCL solver so it can cope with previously unattainable problems. Third, if some intermediate inverse problems are solved, a runtime estimation for inverting $i+1$ steps can be calculated, so the corresponding computational resources can be allocated for this purpose. Fourth, in some cases it is not even needed to invert $i+1$ steps directly since solutions of the intermediate inverse problems can be simultaneously preimages of $i+1$ steps.

The main result of the paper is inverting 29-step MD5 and 24-step SHA-1 for the first time, thus making a clear progress in solving two hard computational problems.

In the future we are going to use more sophisticated parametrization algorithms and apply the proposed technique to analyze other cryptographic hash functions.

References

1 Carlos Ansótegui, Josep Pon, Meinolf Sellmann, and Kevin Tierney. Pydgga: Distributed GGA for automatic configuration. In Chu-Min Li and Felip Manyà, editors, *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 11–20. Springer, 2021. `doi:10.1007/978-3-030-80223-3\_2`.

2 Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In Ian P. Gent, editor, *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*, pages 142–157. Springer, 2009. `doi:10.1007/978-3-642-04244-7\_14`.

3 João Araújo, Choiwah Chow, and Mikolás Janota. Symmetries for cube-and-conquer in finite model finding. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming, CP 2023, August 27-31, 2023, Toronto, Canada*, volume 280 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.CP.2023.8`, `doi:10.4230/LIPICS.CP.2023.8`.

4 Emanuele Bellini, Alessandro De Piccoli, Rusydi H. Makarim, Sergio Polese, Lorenzo Riva, and Andrea Visconti. New records of pre-image search of reduced SHA-1 using SAT solvers. In Debasis Giri, Kim-Kwang Raymond Choo, Saminathan Ponnusamy, Weizhi Meng, Sedat Akleylek, and Santi Prasad Maity, editors, *Proceedings of the Seventh International Conference on Mathematics and Computing - ICMC 2021, Shibpur, India*, volume 1412 of *Advances in Intelligent Systems and Computing*, pages 141–151. Springer, 2021. `doi:10.1007/978-981-16-6890-6\_11`.

5 Armin Biere and Mathias Fleury. Gimsatul, IsaSAT and Kissat entering the SAT Competition 2022. In Tomas Balyo, Marijn Heule, Markus Iser, Matti Järvisalo, and Martin Suda, editors, *SAT Competition 2022 – Solver and Benchmark Descriptions*, volume B-2022-1 of *Department of Computer Science Series of Publications B*, pages 10–11. University of Helsinki, 2022.

6 Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2021. `doi:10.3233/FAIA336`.

7 Curtis Bright, Kevin K. H. Cheung, Brett Stevens, Ilias S. Kotsireas, and Vijay Ganesh. A sat-based resolution of lam's problem. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Event, February 2-9, 2021*, pages 3669–3676. AAAI Press, 2021. URL: `https://doi.org/10.1609/aaai.v35i5.16483`, `doi:10.1609/AAAI.V35I5.16483`.

8 Davin Choo, Mate Soos, Kian Ming Adam Chai, and Kuldeep S. Meel. Bosphorus: Bridging ANF and CNF solvers. In Jürgen Teich and Franco Fummi, editors, *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*, pages 468–473. IEEE, 2019. `doi:10.23919/DATE.2019.8715061`.

9 Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1989. `doi:10.1007/0-387-34805-0\_39`.

10 Debapratim De, Abishek Kumarasubramanian, and Ramarathnam Venkatesan. Inversion attacks on secure hash functions using satsolvers. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing - SAT 2007, 10th International Conference, Lisbon, Portugal, May 28-31, 2007, Proceedings*, volume 4501 of *Lecture Notes in Computer Science*, pages 377–382. Springer, 2007. `doi:10.1007/978-3-540-72788-0\_36`.

11 Donald E. Eastlake and Paul E. Jones. US secure hash algorithm 1 (SHA1). *RFC*, 3174:1–22, 2001. `doi:10.17487/RFC3174`.

12 Thomas Espitau, Pierre-Alain Fouque, and Pierre Karpman. Higher-order differential meet-in-the-middle preimage attacks on SHA-1 and BLAKE. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual*

Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I, volume 9215 of *Lecture Notes in Computer Science*, pages 683–701. Springer, 2015. doi:10.1007/978-3-662-47989-6_33.

13 Marijn Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In Kerstin Eder, João Lourenço, and Onn Shehory, editors, *Hardware and Software: Verification and Testing - 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers*, volume 7261 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2011. doi:10.1007/978-3-642-34188-5_8.

14 Marijn Heule and Hans van Maaren. March_dl: Adding adaptive heuristics and a new branching strategy. *J. Satisf. Boolean Model. Comput.*, 2(1-4):47–59, 2006. URL: https://doi.org/10.3233/sat190016, doi:10.3233/SAT190016.

15 Marijn J. H. Heule. Schur number five. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6598–6606. AAAI Press, 2018. URL: https://doi.org/10.1609/aaai.v32i1.12209, doi:10.1609/AAAI.V32I1.12209.

16 Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean triples problem via Cube-and-Conquer. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 228–245. Springer, 2016. doi:10.1007/978-3-319-40970-2_15.

17 Ekawat Homsirikamol, Pawel Morawiecki, Marcin Rogawski, and Marian Srebrny. Security margin evaluation of SHA-3 contest finalists through sat-based attacks. In Agostino Cortesi, Nabendu Chaki, Khalid Saeed, and Slawomir T. Wierzchon, editors, *Computer Information Systems and Industrial Management - 11th IFIP TC 8 International Conference, CISIM 2012, Venice, Italy, September 26-28, 2012. Proceedings*, volume 7564 of *Lecture Notes in Computer Science*, pages 56–67. Springer, 2012. doi:10.1007/978-3-642-33260-9_4.

18 Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Automated configuration and selection of SAT solvers. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 481–507. IOS Press, 2021. doi:10.3233/FAIA200995.

19 Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In Carlos A. Coello Coello, editor, *Learning and Intelligent Optimization - 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer, 2011. doi:10.1007/978-3-642-25566-3_40.

20 Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 1152–1157. AAAI Press, 2007. URL: http://www.aaai.org/Library/AAAI/2007/aaai07-183.php.

21 Stepan Kochemazov. Exploring the limits of problem-specific adaptations of SAT solvers in SAT-based cryptanalysis. In Leonid Sokolinsky and Mikhail Zymbler, editors, *Parallel Computational Technologies*, pages 149–163. Springer International Publishing, 2021. doi:10.1007/978-3-030-81691-9_11.

22 Florian Legendre, Gilles Dequen, and Michaël Krajecki. Encoding hash functions as a SAT problem. In *IEEE 24th International Conference on Tools with Artificial Intelligence, ICTAI 2012, Athens, Greece, November 7-9, 2012*, pages 916–921. IEEE Computer Society, 2012. doi:10.1109/ICTAI.2012.128.

23 Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization. *J. Mach. Learn. Res.*, 23:54:1–54:9, 2022. URL: http://jmlr.org/papers/v23/21-0888.html.

24 João Marques-Silva and Karem Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, 48(5):506–521, 1999. doi:10.1109/12.769433.

25 Alfred Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996. URL: http://cacr.uwaterloo.ca/hac/, doi:10.1201/9781439821916.

26 Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1989. doi:10.1007/0-387-34805-0_21.

27 Ilya Mironov and Lintao Zhang. Applications of SAT solvers to cryptanalysis of hash functions. In Armin Biere and Carla P. Gomes, editors, *Theory and Applications of Satisfiability Testing - SAT 2006, 9th International Conference, Seattle, WA, USA, August 12-15, 2006, Proceedings*, volume 4121 of *Lecture Notes in Computer Science*, pages 102–115. Springer, 2006. doi:10.1007/11814948_13.

28 Pawel Morawiecki and Marian Srebrny. A SAT-based preimage analysis of reduced Keccak hash functions. *Inf. Process. Lett.*, 113(10-11):392–397, 2013. URL: https://doi.org/10.1016/j.ipl.2013.03.004, doi:10.1016/J.IPL.2013.03.004.

29 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995. URL: https://doi.org/10.1017/cbo9780511814075, doi:10.1017/CBO9780511814075.

30 Heinz Mühlenbein. How genetic algorithms really work: Mutation and hillclimbing. In Reinhard Männer and Bernard Manderick, editors, *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992*, pages 15–26. Elsevier, 1992.

31 Kevin P. Murphy. *Machine learning - a probabilistic perspective.* Adaptive computation and machine learning series. MIT Press, 2012.

32 Saeed Nejati and Vijay Ganesh. CDCL(Crypto) SAT solvers for cryptanalysis. In Tima Pakfetrat, Guy-Vincent Jourdan, Kostas Kontogiannis, and Robert F. Enenkel, editors, *Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, CASCON 2019, Markham, Ontario, Canada, November 4-6, 2019*, pages 311–316. ACM, 2019. URL: https://dl.acm.org/doi/10.5555/3370272.3370307, doi:10.5555/3370272.3370307.

33 Saeed Nejati, Jan Horácek, Catherine H. Gebotys, and Vijay Ganesh. Algebraic fault attack on SHA hash functions using programmatic SAT solvers. In John N. Hooker, editor, *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings*, volume 11008 of *Lecture Notes in Computer Science*, pages 737–754. Springer, 2018. doi:10.1007/978-3-319-98334-9_47.

34 Saeed Nejati, Jia Hui Liang, Catherine H. Gebotys, Krzysztof Czarnecki, and Vijay Ganesh. Adaptive restart and CEGAR-based solver for inverting cryptographic hash functions. In Andrei Paskevich and Thomas Wies, editors, *Verified Software. Theories, Tools, and Experiments - 9th International Conference, VSTTE 2017, Heidelberg, Germany, July 22-23, 2017, Revised Selected Papers*, volume 10712 of *Lecture Notes in Computer Science*, pages 120–131. Springer, 2017. doi:10.1007/978-3-319-72308-2_8.

35 Vegard Nossum. SAT-based preimage attacks on SHA-1. Master's thesis, University of Oslo, Department of Informatics, 2012.

36 Ronald L. Rivest. The MD4 message digest algorithm. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, volume 537 of *Lecture Notes in Computer Science*, pages 303–311. Springer, 1990. doi:10.1007/3-540-38424-3_22.

37 Ronald L. Rivest. The MD5 message-digest algorithm. *RFC*, 1321:1–21, 1992. doi:10.17487/RFC1321.

38 Yu Sasaki, Wataru Komatsubara, Yasuhide Sakai, Lei Wang, Mitsugu Iwamoto, Kazuo Sakiyama, and Kazuo Ohta. Meet-in-the-middle preimage attacks revisited - new results on

MD5 and HAVAL. In Pierangela Samarati, editor, *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*, pages 111–122. SciTePress, 2013. URL: `https://ieeexplore.ieee.org/document/7223160/`.

39 Alexander A. Semenov, Daniil Chivilikhin, Artem Pavlenko, Ilya V. Otpuschennikov, Vladimir Ulyantsev, and Alexey Ignatiev. Evaluating the hardness of SAT instances using evolutionary optimization algorithms. In Laurent D. Michel, editor, *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*, volume 210 of *LIPIcs*, pages 47:1–47:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. URL: `https://doi.org/10.4230/LIPIcs.CP.2021.47`, `doi:10.4230/LIPICS.CP.2021.47`.

40 Alexander A. Semenov, Ilya V. Otpuschennikov, Irina Gribanova, Oleg Zaikin, and Stepan Kochemazov. Translation of algorithmic descriptions of discrete functions to SAT with applications to cryptanalysis problems. *Log. Methods Comput. Sci.*, 16(1), 2020. `doi:10.23638/LMCS-16(1:29)2020`.

41 Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT solvers to cryptographic problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings*, volume 5584 of *Lecture Notes in Computer Science*, pages 244–257. Springer, 2009. `doi:10.1007/978-3-642-02777-2\_24`.

42 Marian Srebrny, Mateusz Srebrny, and Lidia Stepien. SAT as a programming environment for linear algebra and cryptanalysis. In *International Symposium on Artificial Intelligence and Mathematics, ISAIM 2008, Fort Lauderdale, Florida, USA, January 2-4, 2008*, 2008. URL: `http://isaim2008.unl.edu/PAPERS/SS1-AI+Logic/MSrebrny-ss1.pdf`.

43 Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017. `doi:10.1007/978-3-319-63688-7\_19`.

44 Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005. `doi:10.1007/11426639\_2`.

45 Oleg Zaikin. Inverting 43-step MD4 via Cube-and-Conquer. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 1894–1900. ijcai.org, 2022. URL: `https://doi.org/10.24963/ijcai.2022/263`, `doi:10.24963/IJCAI.2022/263`.

46 Oleg Zaikin. Inverting cryptographic hash functions via Cube-and-Conquer. *J. Artif. Intell. Res.*, in press.

47 Oleg Zaikin and Stepan Kochemazov. On black-box optimization in divide-and-conquer SAT solving. *Optim. Methods Softw.*, 36(4):672–696, 2021. `doi:10.1080/10556788.2019.1685993`.